Exam 1
Feb. 23, 25, 27?

- You will be able to use all of the Motorola data manuals on the exam.
- No calculators will be allowed for the exam.
- Numbers
  - Decimal to Hex (signed and unsigned)
  - Hex to Decimal (signed and unsigned)
  - Binary to Hex
  - Hex to Binary
  - Addition and subtraction of fixed-length hex numbers
  - Overflow, Carry, Zero, Negative bits of CCR
- Programming Model
  - Internal registers – A, B, (D = AB), X, Y, SP, PC, CCR
- Addressing Modes and Effective Addresses
  - INH, IMM, DIR, EXT, REL, IDX (Not Indexed Indirect)
  - How to determine effective address
- Instructions
  - What they do - Core Users Guide
  - What machine code is generated
  - How many cycles to execute
  - Effect on CCR
  - Branch instructions – which to use with signed and which with unsigned
- Machine Code
  - Reverse Assembly
- Stack and Stack Pointer
  - What happens to stack and SP for instructions (e.g., PSHX, JSR)
  - How the SP is used in getting to and leaving subroutines
- Assembly Language
  - Be able to read and write simple assembly language program
  - Know basic assembler directives – e.g., equ, dc.b, ds.w
  - Flow charts

Programming the HC12 in C

- A comparison of some assembly language and C constructs

```
Assembly                        |   C
--------------------------------|-----------------------------------
; Use a name instead of a num   |   /* Use a name instead of a num */
COUNT:  EQU  5                  |   #define COUNT 5
;--------------------------------|   /*-----------------------------*/
;start a program                |   /* To start a program */
       org   $2000              |   main()
       lds   #0x2000            |   {
                                |   }
;--------------------------------|   /*-----------------------------*/
```

- Note that in C, the starting location of the program is defined when you compile the program, not in the program itself.
- Note that C always uses the stack, so C automatically loads the stack pointer for you.

```
Assembly                        |   C
--------------------------------|-----------------------------------
;allocate two bytes for        |   /* Allocate two bytes for
;a signed number                |    * a signed number */
                                |
      org     $1000             |
i:    ds.w    1                 |   int i;
j:    dc.w    $1A00             |   int j = 0x1a00;
;--------------------------------|   /*-----------------------------*/
;allocate two bytes for        |   /* Allocate two bytes for
;an unsigned number             |    * an unsigned number */
i:    ds.w    1                 |   unsigned int i;
j     dc.w    $1A00             |   unsigned int j = 0x1a00;
;--------------------------------|   /*-----------------------------*/
;allocate one byte for         |   /* Allocate one byte for
;an signed number               |    * an signed number */
                                |
i:    ds.b    1                 |   signed char i;
j:    dc.b    $1F               |   signed char j = 0x1f;
```

```
Assembly                        |   C
;-----------------------------|   /*-----------------------------*/
;Get a value from an address  |   /* Get a value from an address */
; Put contents of address     |   /*  Put contents of address    */
; $E000 into variable i       |   /*  0xE000 into variable i     */
                                |
i:  ds.b   1                    |   unsigned char i;
                                |
    ldaa   $E000                |   i = * (unsigned char *) 0xE000;
    staa   i                    |
                                |   /*-----------------------------*/
                                |   /* Use a variable as a pointer
                                |      (address) */
                                |
                                |   unsigned char *ptr, i;
                                |
                                |   ptr = (unsigned char *) 0xE000;
                                |   i = *ptr;
                                |   *ptr = 0x55;
;-----------------------------|   /*-----------------------------*/
```

- In C, the construct `*(num)` says to treat `num` as an address, and to work with the contents of that address.
- Because C does not know how many bytes from that address you want to work with, you need to tell C how many bytes you want to work with. You also have to tell C whether you want to treat the data as signed or unsigned.
  - `i = * (unsigned char *) 0xE000;` tells C to take one byte from address `0xE000`, treat it as unsigned, and store that value in variable `i`.
  - `j = * (int *) 0xE000;` tells C to take two bytes from address `0xE000`, treat it as signed, and store that value in variable `j`.
  - `* (char *) 0xE000 = 0xaa;` tells C to write the number `0xaa` to a single byte at addess `0xE000`.
  - `* (int *) 0xE000 = 0xaa;` tells C to write the number `0x00aa` to two bytes starting at addess `0xE000`.

```
Assembly                        |   C
;-----------------------------|   /*-----------------------------*/
;To call a subroutine          |   /* To call a function */
  ldaa  i                       |   sqrt(i);
  jsr   sqrt                    |
;-----------------------------|   /*-----------------------------*/
;To return from a subroutine   |   /* To return from a function */
  ldaa  j                       |   return j;
  rts                           |

;-----------------------------|   /*-----------------------------*/
;Flow control                  |   /* Flow control */
    blo                         |   if (i < j)
    blt                         |   if (i < j)
                                |
    bhs                         |   if (i >= j)
    bge                         |   if (i >= j)
;-----------------------------|   /*-----------------------------*/
                                |
```

- Here is a simple program written in C and assembly. It simply divides 16 by 2. It does the division in a function.

```
ASSEMBLY                    | C
----------------------------|---------------------
                            |
      org     $1000         |  signed char i;
i:    ds.b    1             |
                            |
                            |  signed char div(signed char j);
      org     $1000         |  main()
      lds     #$2000        |  {
      ldaa    #16           |     i = div(16);
      jsr     div           |  }
      staa    i             |
      swi                   |
                            |
div:  asra                  |  signed char div(signed char j)
      rts                   |  {
                            |     return j >> 1;
                            |  }
```

A simple C program and how to compile it

Here is a simple C program

```
#define COUNT 5

unsigned int i;

main()
{
    i = COUNT;
}
```

Details of compiling a program are discussed in detail in the text in Section 5.10. Here is an outline of the details:

1. Open the Embedded GNU (EGNU) IDE.
2. From the **File** menu, select the **New Source File** option. Type in your C program. Then from the **File** menu, select the **Save unit as** submenu, and save your file with an appropriate name and in an appropriate directory.
3. From the **File** menu, select the **New Project** option. Give the project an appropriate name and an appropriate directory. (Note: the project base name must be different from the C file names.) When the **Project Options** popup dialog appears, click the down arrow below **Hardware Profile**, and select **Dragon12**. Click the **Edit Profile** button, and make sure the following are set:
   - `ioports from 0000, length  400`
   - `eeprom  from  400, length  c00`
   - `data    from 1000, length 1000`
   - `text    from 2000, length 2000`
   - `stack   at   2000`

   Then click the **OK** button.
4. From the **Project** menu, select the **Add to project** option, and, in the pop-up dialog box, select the C file you entered in Step 2.
5. From the **Build** menu, select the **Make** option. Under the **Compiler** window at the bottom of the screen, you will hopefully see the message `No errors or warnings.` If not, you will need to fix the errors.
6. If all went well, you should be able to download the compiled file into the 9S12.

If the name of your project is `Project1.prj`, the compiler will generate a file `Project1.dmp`. Here is some of the output from `Project1.dmp`. There are a couple of things you should note about this output:

- The first thing the C program does is load the stack pointer.
- The `main()` function is the assembly language for the C program you entered.

```
00001000 <_start>:
    1000:   cf 20 00        lds #2000 <_stack>
    1003:   16 20 37        jsr 1037 <__premain>

00001006 <__map_data_section>:
    1006:   ce 20 40        ldx #2040 <_etext>
    1009:   cd 10 00        ldy #1000 <__data_section_start>
    100c:   cc 00 00        ldd #0 <__data_section_size>
    100f:   27 07           beq 1018 <__init_bss_section>

00001011 <Loop>:
    1011:   18 0a 30 70     movb    1,X+, 1,Y+
    1015:   04 34 f9        dbne    D,2011 <Loop>

00002018 <__init_bss_section>:
    1018:   cc 00 01        ldd #1 <__bss_size>
    101b:   27 08           beq 1025 <Done>
    101d:   ce 10 00        ldx #1000 <__data_section_start>

00001020 <Loop>:
    1020:   69 30           clr 1,X+
    1022:   04 34 fb        dbne    D,1020 <Loop>

00001025 <Done>:
    1025:   16 20 31        jsr 1031 <main>

00001028 <fatal>:
    1028:   16 20 3b        jsr 103b <_exit>
    102b:   20 fb           bra 1028 <fatal>
    102d:   20 06           bra 1035 <main+0x4>
    102f:   20 18           bra 1049 <_etext+0x9>

00001031 <main>:
    1031:   18 0b 05 10     movb    #5, 1000 <__data_section_start>
    1035:   00
    1036:   3d              rts

00001037 <__premain>:
```

```
    1037:   87              clra
    1038:   b7 02           tap
    103a:   3d              rts

0000103b <_exit>:
    103b:   10 ef           cli
    103d:   3e              wai
    103e:   20 fb           bra 103b <_exit>
```

Pointers in C

- To access a memory location:

  ```
  *address
  ```

- You need to tell compiler whether you want to access 8-bit or 16 bit number, signed or unsigned:

  ```
  *(type *)address
  ```

  - To read from an eight-bit unsigned number at memory location `0x1000`:

    ```
    x = *(unsigned char *)0x1000;
    ```

  - To write an `0xaa55` to a sixteen-bit signed number at memory locations `0x1010` and `0x1011`:

    ```
    *(signed int *)0x1010 = 0xaa55;
    ```

- If there is an address which is used alot:

```
#define PORTB (* (unsigned char *) 0x0001)

x = PORTB;      /* Read from address 0x0001 */
PORTB = 0x55;   /* Write to address 0x0001 */
```

- To access consecutive locations in memory, use a variable as a pointer:

```
unsigned char *ptr;

ptr = (unsigned char *)0x1000;
*ptr = 0xaa;         /* Put 0xaa into address 0x1000 */
ptr = ptr+2;         /* Point two further into table */
x = *ptr;            /* Read from address 0x1002     */
```

- To set aside ten locations for a table:

  ```
  unsigned char table[10];
  ```

- Can access the third element in the table as:

  ```
  table[2]
  ```

  or as

  ```
  *(table+2)
  ```

- To set up a table of constant data:

  ```
  const unsigned char table[] = {0x00,0x01,0x03,0x07,0x0f};
  ```

  This will tell the compiler to place the table of constant data with the program (which might be placed in EEPROM) instead of with regular data (which must be placed in RAM).

- There are a lot of registers (such as `PORTA` and `DDRA`) which you will use when programming in C. Rather than having to define the registers each time you use them, you can include a header file for the HC12 wihch has the registers predefined. Here is the beginning of the header file `iodp256.h`. You can find the complete file on the EE 308 homepage. Here are a few entries from the header file:

```
#ifndef _BASE
#define _BASE   0
#endif
#define _IO(x)  @(_BASE)+(x)
#if _BASE == 0
#define _PORT   @dir
#else
#define _PORT
#endif
#define uint    unsigned int
/*  MEBI Module
 */
_PORT volatile char PORTA     _IO(0x00);   /* port A */
_PORT volatile char PORTB     _IO(0x01);   /* port B */
_PORT volatile char DDRA      _IO(0x02);   /* data direction port A */
_PORT volatile char DDRB      _IO(0x03);   /* data direction port B */
```

Program to make LEDs on Dragon12-Plus board count

```
#include "hcs12.inc"
        lds     #$2000      ; Load stack pointer
        bset    DDRP,#$0F   ; Make PP0-PP3 outputs
        bset    PTP,#$0F    ; Turn off seven-seg LEDs
        bset    DDRJ,#$02   ; Make PJ1 output
        bclr    PRJ,#$02    ; Turn on individual LEDs
        movb    #$FF,DDRB   ; Activate control lines for LEDs
loop:   inc     PORTB       ;
        bsr     delay       ; Wait a bit
        bra     loop        ; Repeat
delay:  ldy     #100
l1:     ldx     #5000
l2:     dbne    x,l2
        dbne    y,l1
        rts
-----------------------------------------------------------------------
#include "hcs12.h"
#define D_1MS (24000/5)   // Inner loop is 5 cycles
#define TRUE 1
void delay(unsigned int ms);

int main()
{
    DDRP = DDRP | 0x0F;     /* Make PP0-PP3 outputs            */
    PTP = PTP | 0x0F;       /* Turn off seven-seg LEDs         */
    DDRJ = DDRJ | 0x02;     /*Make PJ1 output                  */
    PTJ = PTJ & ~0x02;      /* Turn on individual LEDs         */
    DDRB = 0xFF;            /* Activate control lines for LEDs */
    while (TRUE) {          /* Repeat forever                  */
        PORTB = PORTB + 1;  /* Increment LEDs;                 */
        delay(100);         /* Wait a bit                      */
    }
}
void delay (unsigned int ms)
{
    unsigned int i;
    while (ms > 0) {
        i = D_1MS;
        while (i > 0) {
            i = i - 1;
        }
        ms = ms - 1;
    }
}
```

```
Lab03_p1.elf:     file format elf32-m68hc12
Lab03_p1.elf
architecture: m68hc12, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00002000

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000006e  00002000  00002000  00001000  2**0

Disassembly of section .text:

00002000 <_start>:
    2000:   cf 20 00        lds #2000 <_start>
    2003:   16 20 64        jsr 2064 <__premain>

00002006 <__map_data_section>:
    2006:   ce 20 6e        ldx #206e <__data_image>
    2009:   cd 10 00        ldy #1000 <__data_section_start>
    200c:   cc 00 00        ldd #0 <__bss_size>
    200f:   27 07           beq 2018 <__init_bss_section>

00002011 <Loop>:
    2011:   18 0a 30 70     movb    1,X+, 1,Y+
    2015:   04 34 f9        dbne    D,2011 <Loop>

00002018 <__init_bss_section>:
    2018:   cc 00 00        ldd #0 <__bss_size>
    201b:   27 08           beq 2025 <Done>
    201d:   ce 10 00        ldx #1000 <__data_section_start>

00002020 <Loop>:
    2020:   69 30           clr 1,X+
    2022:   04 34 fb        dbne    D,2020 <Loop>

00002025 <Done>:
    2025:   16 20 31        jsr 2031 <main>

00002028 <fatal>:
    2028:   16 20 68        jsr 2068 <_exit>
    202b:   20 fb           bra 2028 <fatal>
    202d:   20 06           bra 2035 <main+0x4>
    202f:   20 18           bra 2049 <main+0x18>
```

```
00002031 <main>:
    2031:   1c 02 5a 0f     bset    25a <__bss_size+0x25a> #$0f
    2035:   1c 02 58 0f     bset    258 <__bss_size+0x258> #$0f
    2039:   1c 02 6a 02     bset    26a <__bss_size+0x26a> #$02
    203d:   1d 02 68 02     bclr    268 <__bss_size+0x268> #$02
    2041:   18 0b ff 00     movb    #255, 3 <__bss_size+0x3>
    2045:   03
    2046:   f6 00 01        ldab    1 <__bss_size+0x1>
    2049:   52              incb
    204a:   7b 00 01        stab    1 <__bss_size+0x1>
    204d:   cc 00 64        ldd #64 <__bss_size+0x64>
    2050:   07 02           bsr 2054 <delay>
    2052:   20 f2           bra 2046 <main+0x15>

00002054 <delay>:
    2054:   04 44 0c        tbeq    D,2063 <delay+0xf>
    2057:   ce 12 c0        ldx #12c0 <__data_section_start+0x2c0>
    205a:   1a e1 e7        leax    -25,X
    205d:   04 75 fa        tbne    X,205a <delay+0x6>
    2060:   04 34 f4        dbne    D,2057 <delay+0x3>
    2063:   3d              rts

00002064 <__premain>:
    2064:   87              clra
    2065:   b7 02           tap
    2067:   3d              rts

00002068 <_exit>:
    2068:   10 ef           cli
    206a:   3e              wai
    206b:   20 fb           bra 2068 <_exit>

0000206d <_etext>:
    206d:   a7              nop
```